2进程通信、线程

2018年10月17日 13:54

-
- 进程通信

1. 互斥与同步也可视作低级进程通信
   1) 效率低，一次只能从缓冲区取得一个消息
   2) 对用户不透明：OS只提供共享存储器，共享数据结构的设置、数据的传送、进程的互斥与同步必须由程序员实现
2. OS提供的高级通信工具
   1) 使用方便，对用户透明：由OS提供一组隐藏具体细节的通信命令/原语，减少了编程复杂性
   2) 高效：高级通信命令/原语可高效传送大量数据

一. 进程通信的类型

1. 共享存储器系统(Shared-Memory System)：通过某些数据结构或共享存储区进行通信
   1) 基于共享数据结构的通信方式
      (1) 操作系统提供共享存储器，程序员负责设置共用数据结构和处理进程间同步
      (2) 仅适于少量数据，效率低，是低级通信
   2) 基于共享存储区的通信方式
      (1) 内存中划出一块共享区，由进程负责访问控制和数据形式、位置
      (2) 通信前，进程先向系统申请共享存储区中的一个分区，得到关键字、描述符，附加到自己地址空间，读写完成后归还共享存储区
      (3) 由进程控制访问和数据形式、位置。是高级通信
2. 管道pipe通信系统
   1) 管道：用于连接读进程和写进程并实现通信的一个共享文件
   2) 以字符流形式，有效传递大量数据，首创于UNIX
   3) 需要这三方面协调能力：
      (1) 互斥：不能同时读写
      (2) 同步：管道有大量数据后，写进程开始睡眠等待，唤醒读进程；同理管道空以后
      (3) 确定对方是否存在：两者都确认存在时才能通信
3. 消息传递系统(Message passing system)
   1) 以格式化的消息(message)为单位交换数据
   2) 数据全部封装在信息中，利用OS提供的通信命令/原语进行传递
   3) **透明化：通信实现细节被隐藏**，降低了通信程序设计复杂性和错误率
   4) 微内核操作系统无一例外用它与服务器通信；多处理机系统，分布式系统，计算机网络中主要通信工具，计算机网络中，称 message 为报文
   5) 是一种高级通信方式，有两种实现：
      (1) 直接通信：利用OS提供的发送原语，直接发给目标进程
      (2) 间接通信：通过共享中间实体（邮箱）
4. 客户机-服务器系统(Client-Server system)
      (1) 是网络环境的各种应用领域的主流通信机制
   1) 套接字(Socket)：通信标识类型的数据结构
      (1) 起源于20世纪70年代的BSD UNIX
      (2) 包含了通信目的地地址、通信使用的端口号、通信网络传输层协议、进程所在网络地址、

对客户或服务器程序提供的不同系统调用/API函数。是进程通信和网络通信的基本构件

i. 基于文件型：同一机器中，套接字关联到一个本地文件系统，类似管道

ii. 基于网络型：非对称方式（发送者提供接受者命名）不同主机的网络环境下，一对套接字分别属于接受进程/服务器端，一个属于发送进程/客户端。一般客户端发出连接请求后，随机申请一个套接字，主机分配一个专用端口绑定该套接字。服务器端拥有全局公认的套接字和指定接口（如ftp服务器监听端口为21，Web或http的为80）服务器一旦从监听接口收到请求，就立刻接受连接，并发送数据，通信结束后，系统关闭服务器端的套接字套接字，撤销连接

(3) 优势是适用于网络环境，而且每个套接字有唯一的套接字标识符，保证了逻辑链路的唯一性，便于并发传输，采用统一接口，隐藏了通信设施和实现细节

2) 远程过程/函数/方法调用

(1) 调用通信协议(Remote Procedure Call)，使运行于本地主机系统上的进程调用另一台远程主机系统上的进程。程序员只需调用该过程，无需为调用过程编程

(2) 负责处理该过程的进程有两个，是本地客户进程和远程服务器进程，他们也称为网络守护进程，主要负责网络间消息传递，一般都处于阻塞状态，等待消息

(3) 为了使该过程调用透明，引入了存根stub的概念，本地每个可独立运行的远程过程都拥有一个客户存根，本地进程调用远程过程实际上是调用该过程关联的存根；服务器端每个进程也存在服务器存根，这些存根一般也处于阻塞状态，等待消息

(4) 调用远程过程的主要步骤是：本地过程调用者调用远程过程在本地关联的客户存根，传递参数，控制权转给客户存根、客户存根建立包括过程名和调用参数等信息的消息，控制权转给本地客户进程、本地客户进程将消息发送给远程服务器进程、远程服务器将消息转给该消息中的远程过程名对应的存根、服务器存根由阻塞转为执行态，拆开消息，取出参数，调用服务器上关联的过程、服务器端远程过程执行完毕后讲结果返回给关联的服务器存根、服务器存根活动控制权，将结果打包成消息，控制权转移给远程服务器进程、远程服务器进程将消息发送回客户端、本地客户进程接收到消息后将消息存入消息中的过程名对应的客户存根，并将控制权转移给客户存根、客户存放取出结果，返回给本地调用者，并转移控制权。

(5) 即：客户过程的本地调用转化为客户存根，再转化为服务器过程的本地调用，客户和服务器不可见中间步骤

二. 消息传递通信的实现方式

1. 直接消息传递系统

(1) 直接通信方式：发送进程利用OS所提供的发送命令/原语，直接把消息发送给目标进程

1) 直接通信原语

(1) 对称寻址方式：显示提供对方标识符

i. send(receiver,message)；receive(sender,message);

ii. 一旦改变进程的名称，需要检查所有进程，不利于进程模块化

(2) 非对称寻址方式：接收原语中只填写源进程的参数，即通信结束后的返回值，发送不变

i. send(receiver,message)；receive(id,message);

ii. id变量可以设置为发送进程的id或名字

2) 消息的格式

(1) 定长消息：适用于单机系统，减少处理和存储消息的开销，也可用于办公系统的便笺通信

(2) 变长消息：可变长度的消息方便了用户，但可能增加了开销

3) 进程的同步方式

(1) 完成消息发送/接收后有这三种状态

i. 发送接收进程都阻塞：实现进程间紧密同步，无缓冲

ii. 只有接收进程阻塞：为了尽快发更多消息给更多目标，应用最广

iii. 都不阻塞：消息发送/接收意外时间各忙各的，也较常见

4) 通信链路（链路可连接多个结点）

(1) 显式/隐式建立

i. 通信前显式调用建立连接命令/原语，请求系统建立，使用完后拆除，适用于网络

ii. 利用系统的发送命令/原语后系统自动隐式建立链路，主要用于单机系统

(2) 通信方式

i. 单向通信链路：只允许一方发，一方收

ii. 双向通信链路：互相收发

2. 信箱通信

(1) 间接通信方式：通过某种中间实体（如共享数据结构）完成，称该实体为邮箱/信箱，有唯

一标识符。信箱建立在随机存储器的共用缓冲区上，暂存消息。信箱只允许核准用户读取

1) 信箱的数据结构
   (1) 信箱头：存放信箱标识符、拥有者、口令、空格数
   (2) 新箱体：若干存放消息或消息头的信箱格，其数目及大小在创建信箱时确定
2) 信箱通信原语
   (1) 创建和撤销，应有创建者给出名字，属性（公用、私用、共享），共享邮箱需给出共享者
   (2) 发送和接收：Send(mailbox,message); Receive(mailbox,message);
3) 信箱属性
   (1) 私用：用户进程自己建立，是该进程的一部分，会随进程结束而消失。其他用户只能发消息，即单向链路
   (2) 公用：由系统创建，在系统运行期间始终存在。所有系统核准的进程都可用它收发消息，即双向链路
   (3) 共享：进程创建，指明给哪些进程名共用，拥有者和共享者都可收发
4) 四种收发关系
   (1) 一对一：专用通信链路，不熟其他进程干扰
   (2) 多对一：服务进程与多用户进程交互，即客户/服务器交互(client/server interaction)
   (3) 一对多：可用广播方式向多个接收者发消息
   (4) 多对多：用公用邮箱互相收发

三. 直接消息传递系统实例

1) 消息缓冲队列通信机制首先由美国的 Hansan 提出，并在 RC 4000 系统上实现

1. 消息缓冲队列通信机制的数据结构

1) 消息缓冲区

```
typedef struct m{
        int sender; //发送者进程标识符
        int size; //消息长度
        char*text;  //消息正文
        struct message_buffer*next; //指向下一个消息缓冲区的指针
        }message_buffer;
```

2) PCB中有关通信的新增数据项

```
struct message_buffer*mq; //消息队列首指针
semaphore mutex; //消息队列互斥信号量
semaphore sm; //消息队列资源信号量
```

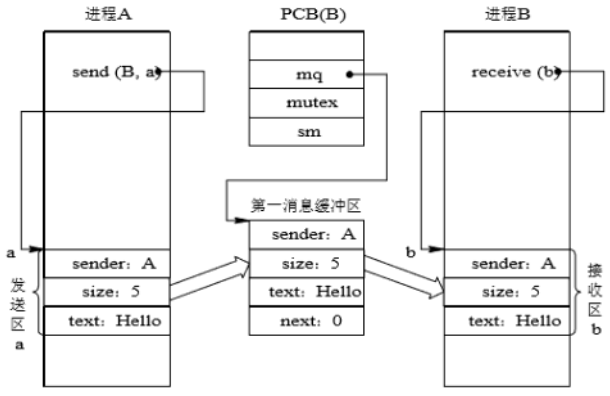

图 2-14 消息缓冲通信

3) 发送原语

1) 利用发送原语发送消息前，应先在内存空间设置发送区a，把消息填入，再调用发送原语，发送原语根据消息长度申请缓冲区i，把a的内容复制到i，再活动接受进程标识符j，把i挂在j的消息队列指针，该队列是临界资源
2) void send(receiver,a){ //向receiver标识符对应进程发送a区的消息

```
getbuf(a.size,i); //申请大小\a.size的名为i的缓冲区
i.sender=a.sender;
i.size=a.size;
strcpy(i.text,a.text);
i.next=0; //用a区的内容初始化i区的内容
getid(PCBset,receiver.j); //找到接收进程的内部标识符
wait(j.mutex);
        insert(j.mq,i); //插到消息队列上
```

signal(j.mutex);
signal(j.sm);}

4) 接收原语

1) 把消息队列上第一个消息缓冲区i的内容复制到消息接收区b内

2) void receive(b){

j=internal name; //接收进程内部标识符
wait(j.sm);
wait(j.mutex);
remove(j.mq,i); //取出第一个消息
signal(j.mutex);
ibsender=i.sender;
b.size=i.size;
strcpy(b.text,i.text); //将i的信息复制到b
releasebuf(i);} //释放缓冲区i

-
- 线程的基本概念

1. 20 世纪 60 年代提出进程的概念后，一直都以它作为拥有资源和独立运行的基本单位。直到80 年代中期，才提出了更小独立运行的基本单位——线程(Threads)。 90 年代后，多处理机系统得到迅速发展，线程能更好提高并行执行程度，充分发挥多处理机的优越性，因而在近几年所推出的多处理机 OS 中都引入了线程

一. 线程的引入

1) 线程减少了程序在并发执行时所付出的时空开销

1. 进程的两个基本属性
   1) 可拥有资源的独立单位
   2) 可独立调度和分派的基本单位
2. 程序并发执行所需的时空开销
   1) 创建进程：分配必需的、除处理机以外的所有资源，如内存空间、I/O 设备，建立相应的 PCB
   2) 撤消进程：资源回收，然后再撤消 PCB
   3) 进程切换：保留当前进程的 CPU 环境，设置新选中进程的 CPU 环境， 花费不少的处理机时间
3. 线程作为调度和分派的基本单位
   1) 希望将进程的两个基本属性分开，由OS分开处理，对拥有资源的基本单位，不施以频繁的切换，因而提出了线程
   2) VLSI技术和计算机体系结构的发展出现了对称多处理机SMP系统，提供了良好的硬件基础
   3) 线程作为调度和分派的基本单位，有效地改善了多处理机系统的性能

二. 线程和进程的比较

1) 通常一个进程都拥有若干个线程，至少也有一个线程

1. 调度的基本单位：线程作为调度和分派的基本单位，而进程作为资源拥有的基本单位；线程切换仅需保存和设置少量寄存器内容，切换代价远低于进程
2. 并发性：所有线程都和进程一样能并发，有效提高资源利用率和系统吞吐量。如文字图形、读入数据、拼写检查放在不同线程；如一个线程专门用于监听客户请求，及时创建其他线程来处理请求
3. 无系统资源：只有TCB、程序计数器、用于保存局部变量、状态参数、和返回地址的寄存器和堆栈
4. 无独立性：每个进程都拥有独立地址空间和其他资源，除了共享全局变量外，不允许其他进程访问；而线程往往是共享进程的内存地址空间和资源
5. 系统开销小：进程创建和撤销除了PCB以外还有各种资源要分配和回收、切换上下文也比线程慢、同步和通信也没线程快，某些OS里线程的切换、同步和通信都无需内核干预
6. 支持多处理机系统：传统进程只能运行在一个处理机上；有多线程的进程能在多个处理机上并行

三. 线程的状态和控制块

1. 三个运行状态：执行：已获得处理机；就绪：具备CPU以外执行条件；阻塞：执行时因某事件而暂停
2. 线程控制块TCB：线程唯一标识符；寄存器：程序计数器PC、状态寄存器、通用寄存器的内容；运行状态；优先级；存储区：切换时的现场信息、线程相关统计信息；信号屏蔽；堆栈指针：用户态和核心态的两套函数调用的局部变量、返回地址的堆栈的指针；

3. 多线程OS中的进程
    1) 拥有资源的基本单位：用户地址空间、同步通信机制的空间、文件、设备、地址映射表
    2) 并发性：线程都属于进程，线程可并发执行
    3) 不是可执行实体：进程执行是指进程某些线程在执行。进程的挂起等状态操作是对所有线程进行

-
- 线程的实现

单进程和单线程系统（DOS）
多进程和单线程系统（UNIX）
单进程和多线程系统（Java Run-time System）
多进程和多线程系统（Windows2000、Solaris、Mach）

一. 线程的实现方式

1. 内核支持线程(Kernel Supported Threads)

1) KST与进程一样与内核紧密相关，TCB存储在内核空间，便于内核感知和控制它
2) 调度以线程为单位，轮转调度算法中线程多的进程可获得更多运行时间
3) 优点：多处理机中并行、不怕阻塞、切换快开销小、内核本身也多线程
4) 缺点：要在用户态到核心态不断切换，因为线程调度和管理靠内核实现

2. 用户级线程(User Level Threads)

1) 仅存在于用户空间，无需内核支持，TCB也在用户空间，内核无法感知
2) 调度以进程为单位，轮转调度算法中多线程进程中的线程只能获得很少运行时间
3) 优点：全程用户态，不用切换、调度算法可由进程自己选择，与OS低级调度算法无关、管理方式是程序代码一部分，无线程平台也可以实现
4) 缺点：进程阻塞使所有线程阻塞、只能单CPU

3. 组合方式
    1) 通过时分多路复用KST实现：KST对应多个ULT。程序员可自由调整KST数目
    2) 三种对应连接方式：
        (1) 多对一模型：多个ULT，仅当需要访问内核时，映射到一个KST，每次只允许射一个
            i. 管理开销小，效率高；但KST阻塞了就会阻塞整个进程，不支持多处理器
        (2) 一对一模型：每个ULT对应一个KST
            i. 不怕阻塞，更好的并发，支持多处理器；但开销极大，KST数是需要限制的
        (3) 多对多模型：许多ULT映射到同样数量或更少数量的KST
            i. 可并行，可多处理器，开销少，效率高，不怕阻塞

二. 线程的实现

1. 无论线程和进程都直接间接需要内核支持
2. 内核支持线程的实现
    1) 创建一个新进程时，便为它在内核中分配一个任务数据区 PTDA(Per Task Data Area)，存储若干 TCB
    2) 只要线程数目未超过系统允许值(通常为数百个)，系统可再为之配新的 TCB 空间；在撤消一个线程时，有时为了节省开销，不收回资源和TCB，方便新进程直接利用
3. 用户级线程的实现
    1) 运行时系统(Runtime System)：管理和控制线程的函数/过程的集合
        (1) 切换时不用转入核心态，而是由运行时系统的线程切换函数将CPU状态保存在该现场的堆栈，调度其他线程，将新线程的CPU状态装入对应CPU寄存器，再切换栈指针和程序计数器，开始新程序运行，这种切换无需进入内核，因而很快
        (2) 资源总是内核管理的。进程通过系统调用，它通过软中断机制（如trap）进入内核，由内核完成资源分配；运行时系统则负责接收线程的资源请求，再通过相关系统调用分配资源
    2) 内核控制线程/轻型进程 LWP(Light Weight Process)
        (1) 每个LWP都有自己的数据结构，可以共享进程的资源，通过系统调用获得内核提供的服务
        (2) LWP相当于缓冲池，又称线程池
        (3) 只有连接到LWP上的线程才能与内核通信，且内核只能看到LWP（连接方式见一.3）
        (4) 内核级线程阻塞后，与之相连的多个LWP也将随之阻塞，与LWP相连的多个用户级线程也阻塞，不过其他LWP仍能执行。LWP阻塞后，线程不能访问内核，但其他操作仍能执行。

类似于进程执行系统调用时，进程会阻塞

(5)

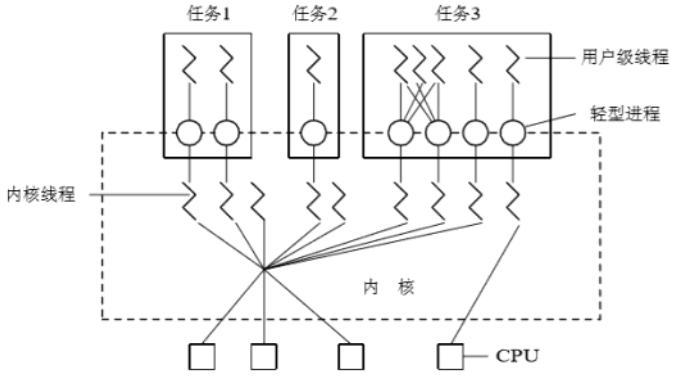

图 2-16 利用轻型进程作为中间系统

三. 线程的创建和终止

1) 线程也由创建而产生，由调度而执行，由终止而消亡

1. 线程的创建
   1) 通常进程刚启动时仅有一个线程，用于创建其他进程
   2) 调用线程创建函数/系统调用，提供参数：程序入口指针、堆栈大小、调度优先级
   3) 创建完返回线程标识符
2. 线程的终止
   1) 调用相应的函数/系统调用来终止
   2) 系统线程等线程建立后就会一直运行下去而不被终止
   3) 多数OS的线程终止后不立即释放资源，仅当进程中的其他线程执行了分离函数后，终止的线程才与资源分离，其资源才能被其他线程利用。释放前，调用者线程课调用“等待线程终止”的命令来连接该线程，然后（阻塞等待）到该线程终止时，调用者继续执行

i.

ii.

iii.

iv.

v.

vi.

vii.

viii. --------我是底线----------